27-领域驱动设计：如何从零开始设计一个软件？

你好！我是郑晔。

在前面的内容中，我给你讲了软件设计的各种基本工具。我们学习了程序设计语言，知道怎样把设计更好地落地；学会了各种编程范式，知道了可以用什么样的元素去做设计；我们还学习了设计原则与模式，知道了怎样组合分解出来的各个元素。

工具都有了，理论也武装上了，那么我们该如何实践呢？或者说，我要去分解组合的东西是从哪而来的呢？这就需要你对**设计方法**有一个基本的认知了，也就是说，我们要理解，在真实世界中，解决具体问题是怎样的一个过程。

那从这一讲开始，我们就来谈谈设计方法的话题，一起了解一下设计的基本过程。

首先，你知道哪些设计方法呢？

我知道的一种做法是，有些人一上来会先设计数据库，因为他们觉得，程序就是数据加函数。数据呢，就要存到数据库里，剩下的就是根据需要对数据库表进行增删改查。但是从我们之前的讲解中，你可以看出，这种思路实际上是一种结构化编程的思路。

后来有人就用面向对象的思路，先来找实体，也就是对象，当然这些实体也要有一些能力。最终，这些对象还是要写到数据库里，同样也是要提供增删改查的能力。

你看，这两种做法本质上没什么太大的区别，都是**围绕着数据在做文章**。在业务需求不复杂的年代，围绕数据做文章的做法还能满足开发的要求，但随着软件日益深入到人们日常工作和生活中，软件变得越来越复杂，这种做法就越发显得笨拙了。

对，软件会越来越复杂的。当软件变得复杂起来，如果我们靠着程序员们本能的做法，就会遇到各种问题，所以，很多人探索了不同的做法。

在诸多的探索之中，有一种做法逐渐脱颖而出，它成功地解决业务软件开发中遇到的大部分问题，这就是**领域驱动设计**。虽然它不是万能药，但对大部分人面对的场景而言，它都能够有效地应对。

领域驱动设计

领域驱动设计（Domain Driven Design，简称 DDD），作为一个新的设计方法正式登上历史舞台，是从 Eric Evans 的著作《领域驱动设计》正式出版开始的。

这种设计方法通过使用通用语言，让业务人员加入到设计过程中，拉近了业务人员与开发人员之间的距离，打破了组织的藩篱。同时，还提供了一套标准的建模方法，帮助团队识别业务模型，避免程序员犯下一些低级错误。

按理说，这种设计方法这么好，应该很快流行起来才对。然而真实情况是，很多程序员都不知道 DDD，一个重要的原因就是 Eric 的这本书写得实在不怎么样。要想从中读出味道，你得比较懂DDD，但是，大多数人并不懂，这就是矛盾的地方。所以，DDD在很长一段时间都被埋没了。

不过，后来，随着**微服务**的兴起，人们越发认识到，微服务的难度并不在于将一个系统拆分成若干的服务，

而在于如何有效地划分微服务。这个时候，人们发现，DDD才是最恰当的指引。关于微服务和 DDD 之间的关系，我在《10x 程序员工作法》里已经讲过了，有兴趣的可以去了解一下，这里就不再赘述了。

现在，你已经知道了DDD的好，准备学习DDD 了。但你只要一打开 DDD 的书，一大堆名词就会扑面而来：限界上下文、聚合根、实体、值对象，等等。我该如何下手学习呢？这是摆在每个DDD学习者面前最严峻的问题。

学习 DDD，就要从理解 DDD 的根基入手：通用语言（Ubiquitous Language）和模型驱动的设计（Model-Driven Design），而领域驱动设计的过程，就是建立起通用语言和识别模型的过程。

通用语言

通用语言，就是在业务人员和开发人员之间建立起的一套共有的语言。要知道，在从前的设计方法中，业务人员总是把问题扔过墙头，让开发人员去解决。可是，业务人员说的都是业务名词，比如：产品、订单等等，而开发人员嘴里全是技术，比如：线程、存储等等，二者除了最基础的几个概念之外，其他的内容基本是没法沟通的。所以，一道人为鸿沟就在开发人员和业务人员之间形成了。

我们在第1讲说过，**软件设计是要在问题和解决方案架设一座桥梁，好的设计要更接近问题**。开发人员对解决方案一端简直再熟悉不过了，但是对业务一端理解则通常不够充分。而通用语言所做的事情，就是把开发人员的思考起点拉到了业务上，也就是从问题出发，这就在一定程度上填平了那道人为的鸿沟。

通用语言是什么呢？就是这个业务中有哪些概念以及哪些操作。比如说，我要做一个电商平台，就要有产品、订单的概念。其中，产品就要有上架、下架、修改产品信息等操作，而订单就会有下单、撤单、修改订单等操作。

在业务人员看来，这里说的都是自己擅长的事情，自己就可以有更多的发言权。在开发人员的视角，概念就是一个一个的类，操作就是一个一个的方法，也很好理解。所以，有一套通用语言，双方皆大欢喜。

但是，通用语言是从哪来的呢？也就是说，如何设计通用语言呢？最简单的做法就是让业务人员和开发人员一起，找一块白板，把各种概念都写在上面。然后，双方重新进行分类整理。

这里面的重点是，让业务人员和开发人员在一起。如果只让一方出现，结果又会是原来的样子，因为你没法判断，这里面的语言对方是否听得懂。

这种做法很简单，但通常都不够系统，会存在各种遗漏。所以，有人探索出一种更正式的实践：**事件风暴**（Event Stroming）。

事件风暴是一个工作坊，基本做法就是找一面很宽的墙，上面铺上大白纸，然后，用便利贴把识别出来的概念贴在上面。当然，前提依然是让业务人员和技术人员都参与其中。

这个实践之所以叫作事件风暴，因为它的关注点在于**领域事件**。领域事件是用来记录业务过程中发生过的重要事情，比如，作为电商平台的工作人员，你想知道产品是不是已经上架了，这个领域事件就是产品已上架；作为消费者，你会关心我的订单是不是下成功了，这个领域事件就是订单已下。

人们做了一个动作，都会关心做过这个动作之后的结果，所以，领域事件用的描述方式都是过去式，比如：OrderPlaced。

事件风暴这个工作坊主要分成三步：

- **第一步就是把领域事件识别出来**，这个系统有哪些是人们关心的结果。有了领域事件，下面一个问题是，这些事件是如何产生的，它必然会是某个动作的结果。
- **第二步就是找出这些动作，也就是引发领域事件的命令**。比如：产品已上架是由产品上架这个动作引发的，而订单已下就是由下单这个命令引发的。
- **第三步就是找出与事件和命令相关的实体或聚合**，比如，产品上架就需要有个产品（Product），下单就需要有订单（Order）。

至此，我们已经把最核心的内容找出来了。通常，在工作坊过程中，为了增强趣味性和清晰性，不同的概念会用不同的颜色的便利贴标识出来，比如，领域事件用橙色、命令用蓝色、实体/聚合用黄色等等。

其实，用不同的颜色建模，事件风暴并不是独一份。Peter Coad也曾提出过一种四色建模的方法：

- 粉色表示时标性对象（moment-interval）；
- 黄色表示角色（role）；
- 蓝色表示描述（description）；
- 绿色表示人、地点、物（party/place/thing）。

他还写了一本《彩色UML建模》（Java Modeling in Color with UML）介绍这种方法。我在ThoughtWorks的前同事徐昊按照自己的理解，对这种方法做了一些更新，有兴趣的话，可以去了解一下。

当然，这里的事件风暴，我只是描述了最简单的一个过程。在具体实施的过程中，还会有更多的细节。不过，最重要的还是，让不同角色的参与其中，让知识在所有人的头脑中进行构建，得到一个大家都认同的结果。

模型驱动设计

有了通用语言，接下来就进入模型设计阶段了。虽然有了通用语言，但是业务人员能够帮到开发人员的还是很少，他们只能告诉开发人员哪些模型是符合业务概念的。

但这么多的业务模型，该如何组织呢？怎样补全欠缺的模型，使之成为一个可以落地的方案呢？这就是开发人员要想办法解决的事情了。

也正是因为在通常情况下，业务模型数量众多，所以在 DDD 的过程中，我们将设计分成了两个阶段：**战略设计**（Strategic Design）和**战术设计**（Tactical Design）。

战略设计是高层设计，是指将系统拆分成不同的领域。而领域驱动设计，核心的概念就是领域，也就是说，它给了我们一个拆分系统的新视角：按业务领域拆分。

比如，我把一个电商系统拆分成产品域、订单域、支付域、物流域等。拆分成领域之后，我们识别出来的各种业务对象就会归结到各个领域之中。然而，有时候，不同领域的业务对象会进行交互，比如，我要知道自己订单的物流情况。所以，要在不同的领域之间设计一些交互的方式。

而战术设计是低层设计，也就是如何具体地组织不同的业务模型。在这个层次上，DDD 给我们提供了一些

标准的做法供我们参考。比如，哪种模型应该设计成实体，哪些应该设计成值对象。

我们还要考虑模型之间是什么样的关系，比如，哪些模型要一起使用，可以成为一个聚合。接下来，我们还需要考虑这些模型从哪来、怎样演变，DDD 同样为我们提供了一些标准的设计概念，比如仓库、服务等等。

通过战略设计和战术设计，我们就可以把发现出来的不同业务概念各归其位了。

总结时刻

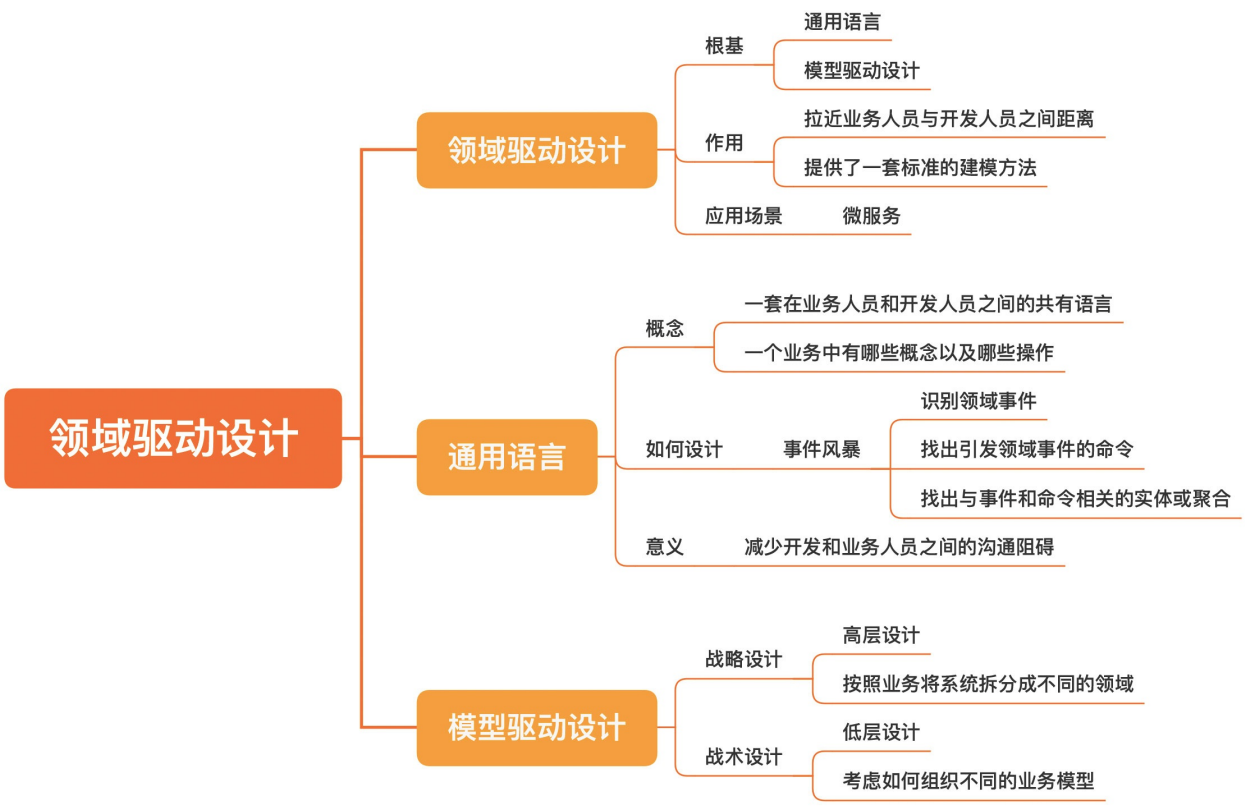
今天，我们讲了领域驱动设计，这是目前在软件行业内最符合软件发展趋势的一种设计方法，因为它把软件设计的起始点从技术拉到了业务。

学习领域驱动设计，我们要从通用语言和模型驱动设计入手。通用语言是在业务人员和技术人员之间建立一套共有的语言，开发通用语言的一种实践是事件风暴，这是一种工作坊，通过识别领域事件找到引发事件的命令，找出与事件和命令相关的实体或聚合，帮助团队建立通用语言。

DDD 的模型设计可以分为战略设计和战术设计。战略设计是高层设计，将系统拆分成领域，战术设计是低层设计，考虑如何组织不同的模型。

好，我们已经对 DDD 有了一个初步的了解。接下来的两讲，我们就分别来看看，如何进行战略设计和战术设计。

如果今天的内容你只能记住一件事，那请记住：**建立一套业务人员和开发人员共享的通用语言**。



思考题

最后，我想请你分享一下，你们在实际工作中是如何与业务人员沟通的？欢迎在留言区写下你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

精选留言：

- 阳仔 2020-07-29 08:10:45
  总结一下
  理解DDD就要理解通用语言和模型驱动设计。
  通用语言就是要把业务人员和开发人员对具体业务概念和逻辑的理解达成一致，可使用事件风暴和彩色建模等方法建立通用语言
  模型驱动设计可以从战略设计和战术设计两方面入手，战略设计属于高层设计，将系统安装领域拆分；战术设计属于低层设计，考虑的是如何组合业务模型 [3赞]

  作者回复2020-07-29 08:48:45
  总结得好！

- 蓝士钦 2020-07-29 08:51:19
  领域驱动设计可以和传统的面向数据库设计的方式结合吗，比如引入一个model模块用来聚合模型 [1赞]

  作者回复2020-07-29 10:25:54
  先按照一个思路走，否则，你会回到老路上去的。

- jg4igianshu 2020-07-31 07:36:59
  实体：在时间上有连续性，并且有唯一标识可以来区分的对象，具有生命周期和行为。
  值对象：用来描述事物的，不区分谁是谁的，不可变的对象，不具有生命周期和行为。

  作者回复2020-07-31 09:34:41
  学会抢答了😅

- 人间四月天 2020-07-30 08:58:33
  领域事件风暴，使用的是事件，动作，实体，建立基本业务模型，我认为这是时序图，状态图的另外一种表达，战略设计，很像业务架构设计，按照领域职责去划分。
  建立一个好的平台，需要做好职责划分，配置团队，要不就是无尽的折磨。

- Jxin 2020-07-29 12:52:56
  1.各自阐述理解。
  2.消除分歧。
  3.达成共识。

- 小学一年级 2020-07-29 12:28:00
  数据加函数用了好多年，现在终于想通了为啥 EF 有种 codefirst编程模式， 因为别人早就想通了 面向对象编程！！！